НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«До захисту допущено»
В.о. завідувача кафедри
_____ __*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему: *Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud*

Виконав: студент 4 курсу, групи ІП-52

Бурлаченко Євгеній Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. викладач Халус О.А.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц. к.т.н. Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *доц. каф. ТК, к.т.н., доц.Тимошин Ю.А.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут ім. І.Сікорського”

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматизованих систем обробки інформації і управління
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050103 «Програмна інженерія»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Бурлаченко Євгенія Олександровича
(прізвище, ім’я, по батькові)

1. Тема проекту *Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud*

керівник проекту Халус Олена Андріївна, ст. викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету від «23» квітня 2019 р. № 1181-с

2. Термін подання студентом проекту «3» червня *2019 року*

3. Вихідні дані до проекту

Технічне завдання

4. Зміст пояснювальної записки

1.Аналіз вимог до програмного забезпечення
2. Моделювання та конструювання програмного забезпечення
3. Аналіз якості та тестування програмного забезпечення
4. Впровадження та супровід програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових кресленників, плакатів, презентацій тощо)

1. Схема структурна варіантів використання програмного забезпечення
2. Схема структурна бази даних програмного забезпечення
3. Схема структурна бізнес процесу
4. *Креслення вигляду екранних форм*

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» лютого *2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	*Вивчення предметної області*	15.02.2019	
2	*Аналіз існуючих методів розв'язання задачі*	23.03.2019	
3	*Постановка та формалізація задачі*	25.03.2019	
4	*Аналіз вимог до програмного забезпечення*	03.04.2019	
5	*Моделювання програмного забезпечення*	10.04.2019	
6	*Оформлення пояснювальної записки*	21.05.2019	
7	*Подання ДП на попередній захист*	28.05.2019	
8	*Подання ДП рецензенту*	03.05.2019	
9	*Подання ДП на основний захист*	08.06.2019	

Студент _____ (підпис) Бурлаченко Є.О.

Керівник проекту _____ (підпис) Халус О.А.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5201.045430.01.81	Пояснювальна записка	59	
3	А4	КПІ.ІП-5201.045430.02.91	Технічне завдання	11	
4	А4	КПІ.ІП-5201.045430.03.51	Програма та методика тестування	12	
5	А4	КПІ.ІП-5201.045430.04.34	Керівництво користувача.	7	
6	А4	КПІ.ІП-5201.045430.05.33	Керівництво адміністратора.	5	
7	А4	КПІ.ІП-5201.045430.06.99 СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-5201.045430.06.99 СБД	Схема структурна бази даних	1	
9	А3	КПІ.ІП-5201.045430.06.99 СС	Схема структурна бізнес процесу	1	
10	А3	КПІ.ІП-5201.045430.06.99 КЕ	Креслення екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-5201.045430.00.90		
Розробн.		Бурлаченко Є.О.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Халус О.А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-52	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5201.045430.01.81	Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud. Пояснювальна записка	59	
3	А4	КПІ.ІП-5201. 045430.02.91	Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud.Технічне завдання	11	
4	А4	КПІ.ІП-5201.045430.03.51	Програма та методика тестування.	12	
5	А4	КПІ.ІП-5201..045430.04.34	Керівництво користувача.	7	
6	А4	КПІ.ІП-5201.045430.05.33	Керівництво адміністратора.	5	
7	А3	КПІ.ІП-5201.045430.06.99	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-5201.045430.06.99	Схема структурна бази даних	1	
9	А3	КПІ.ІП-5201.045430.06.99	Схема структурна бізнес процесу	1	
10	А3	КПІ.ІП-5201.045430.06.99	Креслення екранних форм	1	

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з трьох розділів, містить 20 таблиць та 11 джерел – загалом 59 сторінок.

Об`єкт дослідження: серверні веб системи, що використовуються для виконання неперервної інтеграції та доставки програмного забезпечення, інтегруючись з сервісами Github та кластером Kubernetes.

Мета дипломного проекту: створити легко розширюваний веб застосунок, здатний до швидкого та частого виконання процесу інтеграції та доставки ПЗ до кластера Kubernetes.

У першому розділі було розроблено архітектуру мікросервісного веб застосунку. Побудовано структурну схему класів та діаграму послідовності.

У другому розділі проведено тестування мікросервісного веб застосунку за розробленим планом тестування. Описано процес тестування.

У третьому розділі описано розгортання та впровадження веб застосунку, а також наведено схему структурну розгортання.

У додатках наведено: опис програми, схема структурна класів програмного забезпечення, схема структурна послідовності виконання.

КЛЮЧОВІ СЛОВА: НЕПЕРЕРВНА ІНТЕГРАЦІЯ, НЕПЕРЕРВНА ДОСТАВКА, МІКРОСЕРВІС

ABSTRACT

Explanatory note of the diploma project consists of 3 sections, 20 tables and 11 sources – total 59 pages.

The object of study: server application systems, which can be used for continuous integration and delivery with Github and Kubernetes integration.

The aim of the diploma project: create easy scalable web application, capable of fast, rapid execution of integration and delivery process to Kubernetes cluster.

In the first section, the architecture of microservice web applications was described and developed. A structural diagram of classes and a sequence diagram are constructed.

In the second secion, resulting web application was tested according to the developed test plan. The process of testing is described.

The third section describes the deployment and implementation of microservice web applications. The diagram of structural deployment is provided.

Annexes contain the description of the program, the diagram of the structural classes of the software, the scheme of the structural sequence of execution.

KEYWORDS: CONTINUOUS INTEGRATION, CONTINUOUS DELIVERY, MICROSERVICES

Пояснювальна записка
до дипломного проекту

на тему: Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud _____
_____

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CI – Continious integration – неперервна інтеграція

CD - Continious delivery – неперервна доставка

Деплой – логічна одиниця, що означає примірник програмного забезпечення, що запущено на певній платформі з певним налаштуванням під конкретний варіант використання (для тестування при розробці, для штатного використання тощо).

Ревізія – версія деплою, що має свої налаштування, версію програмного забезпечення. Використовується для слідкування за розвитком програмного забезпечення та для відновлення роботи деплою на разі проблем з поточною ревізією шляхом відкату деплою до налаштувань та версії програмного забезпечення з останньої робочої ревізії.

Контейнер – віртуалізоване на рівні операційної системи середовище, що дозволяє запускати програмне забезпечення ізольовано від інших контейнерів та операційної системи.

Docker – набір інструментів для роботи з контейнерами операційної системи Linux. Надає зручний інтерфейс керування контейнерами, сценарії налаштування та побудови (відомі як Dockerfile).

Dockerfile – сценарій налаштування та побудови контейнерів Docker. Дозволяє у зручній декларативно-імперативній формі описати всі дані та команди необхідні для побудови робочого образу контейнеру.

Kubernetes (скорочено k8s) – система оркестрації контейнерів у кластері. Надає можливість зручного керування машинами та контейнерами у кластері за допомогою API або утиліти командного рядку kubectl.

Маніфест Kubernetes – декларативний опис ресурсу k8s за допомогою мови розмітки YAML або JSON.

ВСТУП

На даний момент під час розробки з’являються проблеми, пов’язані з необхідністю виконувати однакові, довгі операції такі як тестування, побудова, доставка примірників програмного продукту. Зі зміною методологій розробки, що потребують швидкої розробки нових проміжкових версій програмного забезпечення дані операції можуть виконуватися до декількох разів на день. Зважаючи на те, що дані операції можуть виконуватися на великій кількості окремих елементів програмного забезпечення або просто на різних проектах в одній компанії, а також необхідність повторно їх виконувати (наприклад через знайдені помилки на етапі тестування, помилках компіляції тощо), виникає потреба у створенні системи, що буде виконувати їх максимально автономно, зменшуючи потребу у використанні людського ресурсу. Такі системи вже наявні, але більшість з них є застарілими так як вони не були орієнтовані на роботу з контейнерами та хмарними платформами. Саме тому проблема створення системи розгортання програмних продуктів, орієнтованої на роботу з контейнерами Docker та інтеграцією з хмарною платформою Google Cloud Platform є актуальною.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Основної ціллю є створення системи розгортання програмних продуктів на платформі Google Cloud Platform з використанням сервісу GitHub.

Мета розробки - узагальнення теоретичних знань отриманих впродовж учбового процессу для вирішення проблеми інтеграції та розгортання програмних продуктів.

Призначення розробки - прискорення процесу інтеграції та розгортання нових версій програмних продуктів за допомогою сервісу системи контроля версій git GitHub, на платформі Google Cloud Platform.

Задачі:

- отримання інформації про оновлення кодової бази від сервісу GitHub;
- виконання процесу інтеграції оновленого коду;
- виконання процесу побудови оновленого коду;
- виконання процесу розгортання оновленого програмного продукту на платформі GCP;
- ведення конфігурацій розгортання програмних продуктів.

Цілі:

- надати користувачеві (операційному інженерові або програмісту) інтерфейс для зручного ведення конфігурацій розгортання та інтеграції програмних продуктів;
- надати альтернативу наявних системам CI\CD, працюючим по методу SaaS;
- виконувати процес розгортання оновленого програмного продукту, починаючи с етапу оновлення коду до етапу встановлення та налаштування готових програм у середовищі Kubernetes платформи GCP;
- надати користувачеві зручний формат конфігурації розгортання та інтеграції оновленого програмного продукту.

Програмний продукт, що задовольняє вимогам повинен бути написаний на мові, що дозволяє запуск на операційній системі Linux. Він повинен підтримувати HTTP запити по протоколу REST, з кодуванням данних за допомогою JSON, для надання користувацького API та взаємодії з GitHub. Для підтримки роботи з Docker необхідна можливість комунікації з процесом Docker через Unix socket або за допомогою викликів утиліти командного рядку. Робота з кластером kubernetes виконується за допомогою викликів утиліти kubectl або використанням API за допомогою REST або gRPC протоколу.

Для забезпечення високої доступності та зручності масштабування програмний продукт повинен бути розділений на окремі компоненти, що відповідають за окремі процеси, для чого краще за все використовувати мікросервісний підхід, для обміну інформацією між мікросервісами використовувати gRPC з кодуванням данних у форматі Protobuf.

1.2 Змістовний опис і аналіз предметної області

Комплекси для інтеграції та розгортання програмних продуктів мають підтримувати хоча б одну з популярних хмарних систем контролю версій як GitHub, BitBucket чи GitLab. Вони повинні мати можливість запускати процес одразу після оновлення кодової бази, з поверненням результату побудови у систему контролю версій. Такі комплекси мають також інтегруватися з хмарними рішеннями для зберігання образів контейнерів Docker з результатами побудов для подальшої доставки їх у деплої кластеру Kubernetes. Так як ці процеси можуть виконуватися дуже часто та одночасно, комплекс повинен бути оптимізований під паралельне виконання великої кількості задач та масштабованість.

1.3 Аналіз успішних IT-проектів

Серед систем та сервісів неперервної інтеграції комплекс задач має таких важливих конкурентів як CircleCI та Jenkins

Серед інструментів для автоматизації та поліпшення процесу неперервного розгортання у середовищі Kubernetes даний проект має такого основного конкурента як Helm. Також можна виділити утиліту deploy-to-kube

CircleCI - сервіс неперервної інтеграції з підтримкою GitHub [1]. У порівнянні з комплексом задач постачається у двох видах - хмарному та для власних серверів. Завдяки хмарному постачанні адміністратору не потрібно самостійно встановлювати та налагоджувати CircleCI. На відміну від комплексу задач має більш гнучку систему управління правами користувачів. Але на відміну від CircleCI, даний проект має пряму підтримку та інтеграцію з Kubernetes, тоді як з CircleCI адміністратор повинен сам налагоджувати інтеграцію. Також для комплексу задач не потрібно використовувати додаткові мови опису процесу інтеграції, так як для цього використовується сценарій побудови Docker контейнеру - Dockerfile

Jenkins - програмний набір для неперервної інтеграції [2]. У порівняння з комплексом задач має підтримку дуже великої кількості можливостей та інтеграцій за допомогою сторонніх додатків - плагінів. Але на відміну від даного проекту потребує більш важкого процесу встановлення та налагодження, так як постачається у вигляді головного та дочірніх сервісів, що потрібно спочатку встановити на відповідні машини та об’єднати у єдину систему. Також у порівнянні з комплексом задач, Jenkins використовує складну мову для опису процесу інтеграції, що уповільнює налагодження неперервної інтеграції. Docker та Kubernetes не підтримуються безпосередньо Jenkins, тому робота з ними відбувається через плагіни, що супроводжується додатковим налаштуваннями та можливими проблемами з роботою самих плагінів.

Helm - пакетний менеджер для Kubernetes [3]. За допомогою його можна швидко та зручно встановлювати сторонне та власне ПЗ у кластер за допомогою файлів описання процесу встановлення - чартів. Перевага його над даним проектом у наявності великої кількості готових користувацьких чартів, що допомагають швидко додавати до кластеру базові компоненти як СУБД,

кеші, брокери повідомлень тощо. Мінусом порівняно з комплексом задач є необхідність розуміти структуру чартів для їх написання та підтримки, а також необхідність встановлення Helm усередині кластеру, що потребує виділення додаткових ресурсів.

deploy-node-app - утиліта, що дозволяє автоматизувати процес розгортання Node.js сервісів у кластер Kubernetes [4]. Порівняно з комплексом задач він орієнтований лише на Node.js проекти, не може бути легко інтегрований с системою неперервної інтеграції, має дуже мало опцій для налаштування.

Скорочену порівняльну характеристику розглянутих продуктів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика проектів

Назва Функція	CircleCI	Jenkins	Helm	DNA	Комплекс задач
Інтеграція з GitHub	Вбудована	Плагін	-	-	Вбудована
Неперервна інтеграція	Вбудована	Вбудована	-	-	Вбудована
Підтримка Docker	Вбудована	Плагін	-	Вбудована	Вбудована
Підтримка GCR	За допомогою Docker	За допомогою Docker	За допомогою Docker	За допомогою Docker	Вбудована
Підтримка Kubernetes	Плагін	Плагін	Вбудована	Вбудована	Вбудована
Open source	Ні	Ні	Так	Так	Так

1.4 Розроблення функціональних вимог

В системі передбачено наступні варіанти використання:

Таблиця 1.2 – Варіант використання UC01

Назва	Авторизація адміністратора
Опис	Будь який адміністратор має можлівість авторизуватися у комплексі
Учасники	Адміністратор
Передумови	
Постумови	Адміністратор пройшов авторизацію у комплексі
Основний сценарій	Комплекс демонструє вікно авторизації/регістрації яке містить поле вводу логіну та паролю Адміністратор заповнює поля Адміністратор натискає кнопку “Login” Комплекс авторизує адміністратора
Розширення сценаріїв	Комплекс не може авторизувати користувача через неправильні дані Комплекс демонструє повідомлення про помилку

Таблиця 1.3 – Варіант використання UC02

Назва	Перегляд списку користувачів
Опис	Будь який адміністратор має можлівість перегляду списку користувачів
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі
Постумови	Надано список користувачів
Основний сценарій	Адміністратор надсилає запит на отримання списку користувачів Комплекс відображає список усіх користувачів
Розширення сценаріїв	

Таблиця 1.4 – Варіант використання UC03

Назва	Зміна ролей користувачів
Опис	Будь який адміністратор має можлівість змінити роль користувача
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі
Постумови	Змінено роль користувача
Основний сценарій	Адміністратор надсилає запит на зміну ролі користувача Комплекс змінює роль користувача
Розширення сценаріїв	

Таблиця 1.5 – Варіант використання UC04

Назва	Видалення користувачів
Опис	Будь який адміністратор має можлівість видалити користувача
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі Адміністратор на головній сторінці адміністратора
Постумови	Видалено користувача
Основний сценарій	Адміністратор надсилає запит на видалення користувача Комплекс видаляє користувача
Розширення сценаріїв	

Таблиця 1.6 – Варіант використання UC05

Назва	Перегляд списку деплоїв
Опис	Будь який адміністратор має можлівість перегляду списку деплоїв
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі Адміністратор на головній сторінці адміністратора
Постумови	Надано список деплоїв
Основний сценарій	Адміністратор надсилає запит на отримання списку деплоїв Комплекс відображає список усіх деплоїв
Розширення сценаріїв	

Таблиця 1.7 – Варіант використання UC06

Назва	Редагування деплою
Опис	Будь який адміністратор має можлівість редагувати деплой
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі Адміністратор на головній сторінці адміністратора
Постумови	Змінено роль користувача
Основний сценарій	Адміністратор надсилає запит на редагування деплою Комплекс змінює деплой
Розширення сценаріїв	

Таблиця 1.8 – Варіант використання UC07

Назва	Додання деплою
Опис	Будь який адміністратор має можлівість додати деплой
Учасники	Адміністратор
Передумови	Адміністратор пройшов авторизацію у комплексі
Постумови	Змінено роль користувача
Основний сценарій	Адміністратор переходить на сторінку додання деплою Адміністратор вводить дані для додання деплою, натискає кнопку “Add” Комплекс додає деплой
Розширення сценаріїв	

Таблиця 1.9 – Варіант використання UC08

Назва	Авторизація користувача
Опис	Будь який користувач має можлівість авторизуватися у комплексі
Учасники	Користувач
Передумови	
Постумови	Користувач пройшов авторизацію у комплексі
Основний сценарій	Комплекс демонструє вікно авторизації/регістрації яке містить поле вводу логіну та паролю Користувач заповнює поля Користувач натискає кнопку “Login” Комплекс авторизує користувача
Розширення сценаріїв	Комплекс не може авторизувати користувача через неправильні дані Комплекс демонструє повідомлення про помилку

Таблиця 1.10 – Варіант використання UC09

Назва	Редагування Dockerfile
Опис	Будь який користувач має можлівість редагувати Dockerfile у своєму репозиторію
Учасники	Користувач
Передумови	
Постумови	Змінено зміст Dockerfile
Основний сценарій	Користувач змінює зміст Dockerfile
Розширення сценаріїв	

Таблиця 1.11 – Варіант використання UC10

Назва	Додання репозиторію
Опис	Будь який користувач має можлівість додати репозиторій
Учасники	Користувач
Передумови	Користувач пройшов авторизацію у комплексі
Постумови	Додано репозиторій до комплексу
Основний сценарій	Користувач переходить на сторінку додання репозиторію Користувач вводить дані для додання репозиторію, натискає кнопку “Add” Комплекс додає репозиторій
Розширення сценаріїв	Дані введено невірно Комплекс демонструє користувачу повідомлення про помилку

Таблиця 1.12 – Варіант використання UC11

Назва	Редагування конфігурації репозиторію
Опис	Будь який користувач має можлівість редагувати конфігурацію репозиторій
Учасники	Користувач
Передумови	Користувач пройшов авторизацію у комплексі
Постумови	Змінено конфігурацію репозиторію
Основний сценарій	Користувач переходить на сторінку репозиторію Користувач додає нову конфігурацію репозиторію, натискає кнопку “Add” Комплекс додає репозиторій

Продовження таблиці 1.12

Розширення сценаріїв	Дані введено невірно Комплекс демонструє користувачу повідомлення про помилку

Таблиця 1.13 – Варіант використання UC12

Назва	Видалення репозиторію
Опис	Будь який користувач має можлівість видалити репозиторій
Учасники	Користувач
Передумови	Користувач пройшов авторизацію у комплексі
Постумови	Видалено репозиторій з комплексу
Основний сценарій	Користувач переходить на сторінку репозиторію Користувач натискає кнопку “Delete” Комплекс видаляє репозиторій
Розширення сценаріїв	

Таблиця 1.14 – Варіант використання UC13

Назва	Перегляд списку історій побудов репозиторію
Опис	Будь який користувач має можлівість перегляду списку історій побудов репозиторію
Учасники	Користувач
Передумови	Користувач пройшов авторизацію у комплексі Користувач на сторінці репозиторію
Постумови	Надано список історій побудов репозиторію

Продовження таблиці 1.14

Основний сценарій	Користувач на сторінці репозиторію Комплекс відображає список усіх історій побудов
Розширення сценаріїв	

Таблиця 1.15 – Варіант використання UC14

Назва	Перегляд статусу історії побудові репозиторію
Опис	Будь який користувач має можлівість перегляду статусу історії побудови репозиторію
Учасники	Користувач
Передумови	Користувач пройшов авторизацію у комплексі Користувач на сторінці історії побудови репозиторію
Постумови	Надано статус історії побудов репозиторію
Основний сценарій	Користувач на сторінці статусу історії побудови репозиторію Комплекс відображає статус історії побудови
Розширення сценаріїв	

У таблиці 1.16 наведено функціональні вимоги до компексу задач

Таблиця 1.16 – Функціональні вимоги до комплексу задач

Варіант використання	Опис	Пріорітет
Додання репозиторію	1. Комплекс задач повинен мати можливість виконувати побудову з репозиторіїв Github. 1.2. Повинні підтримуватися як приватні так і публічні репозиторії	Високий
Додання деплою	2. Комплекс задач повинен мати можливість створювати та керувати деплоями у kubernetes	Високий
Перегляд статусу історії побудові репозиторію	3. Комплекс задач повинен надавати користувачу статус побудов репозиторію. 3.1. Він також повинен надавати цей статус до Github.	Середній
Перегляд списку деплоїв	4. Комплекс задач повинен надавати користувачу статус деплоїв у kubernetes.	Середній

У таблиці 1.17 наведено матрицю трассування

Таблиця 1.17 – Матриця трассування

Функціональні вимоги / Варіант використання	Побудова з GitHub	Публічні та приватні репозиторії	Створення у деплоїв у kubernetes	Статус репозіторію	Статус у Github	Статус деплою
Додання репозиторію	■	■				
Додання деплою			■			
Перегляд статусу історії побудові репозиторію				■	■	
Перегляд списку деплоїв						■

1.5 Висновки по розділу

У цьому розділі було описано та проаналізовано предметну область розробки. Було виділено успішні IT- проекти у даній області та виконано порівняння даного комплексу задач с готовими продуктами, де було розглянуто їх переваги та недоліки, виконано порівняння з даним комплексом задач та надано у вигляді таблиці.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Розроблена система виконується під операційними системами на базі ядра Linux. Використання під операційними системами Windows та macOS можливе, але не було протестовано.

Для розробки браузерної частини системи було застосовано мову Typescript, з використанням фреймворку Angular 8. Використано паттерни MVC та Dependency Injection, які дозволяють зручно та швидко додавати новий функціонал та відокремлювати різні частини коду, оскільки MVC дозволяє відокремити дані від бізнес-логіки та від логіки відображення, а Dependency Injection видяляє строгі зв`язки між компонентами, дозволяючи швидко замінити одні компоненти на інші з аналогічним функціоналом. Взаємодія з серверною частиною відбувається по протоколу HTTP REST з кодуванням даних за допомогою JSON.

Для розробки серверної частини було застосовано мову Go. Використано паттерн Dependency Injection. Окремі функціональні компоненти розділені між собою за допомогою мікросервісного підходу. Взаємодія між мікросервісами відбувається по протоколу gRPC з кодуванням даних за допомогою Protobuf, а також за допомогою черги повідомлень RabbitMQ. Даний підхід дозволяє легко підтримувати окремі мікросервіси, зручно та швидко додавати новий функціонал та забезпечує високу надійність та масштабованість системи. Опис інтерфейсів наведено у таблиці 2.1, опис класів (або структур, у випадку з серверною частиною, так як мова Go не має класів) – у таблиці 2.2. Більш детальний опис методів наведено у таблиці 2.3. Структурна схема порядку виконання наведено у окремому документі.

Таблиця 2.1 – Опис інтерфейсів системи

Інтерфейс	Опис
DatabaseClient	Інтерфейс, який використовується для роботи з базою даних
Queue	Інтерфейс, який використовується для роботи з чергою повідомлень
Logger	Інтерфейс, що використовується для логів
DistLock	Інтерфейс, що використовується для роботи з розподіленими блокуваннями

Таблиця 2.2 – Опис класів (структур) системи

Клас (структура)	Опис
AdminDashComponent	Клас, який містить бізнес-логіку головної сторінки адміністратора
UsersListComponent	Клас, який містить бізнес-логіку списку користувачів
BranchConfigsListComponent	Клас, який містить бізнес-логіку компоненту списку налаштувань гілок
BuildComponent	Клас, який містить бізнес-логіку компоненту сторінки керування побудовами
BuildsListComponent	Клас, який містить бізнес-логіку компоненту списку побудов
CreateBranchComponent	Клас, який містить бізнес-логіку компоненту для створення нової конфігурації для гілки

Продовження таблиці 2.2

Клас (структура)	Опис
CreateDeploymentComponent	Клас, який містить бізнес-логіку компоненту для створення нового деплою
CreateRepoComponent	Клас, який містить бізнес-логіку компоненту для додання нового репозиторію
DeploymentComponent	Клас, який містить бізнес-логіку компоненту сторінки деплою
DeploymentsListComponent	Клас, який містить бізнес-логіку компоненту списку деплоїв
LoginComponent	Клас, який містить бізнес-логіку компоненту сторінки авторизації
RepoComponent	Клас, який містить бізнес-логіку компоненту сторінки репозиторію
ReposListComponent	Клас, який містить бізнес-логіку компоненту списку репозиторіїв
RevisionComponent	Клас, який містить бізнес-логіку компоненту сторінки ревізії
RevisionsListComponent	Клас, який містить бізнес-логіку компоненту списку ревізій
ClipperService	Клас, який надає методи для взаємодії з API серверної частини
StorageService	Клас, який надає методи для взаємодії з локальним сховищем браузеру
SettingsComponent	Клас, який містить бізнес-логіку компоненту сторінки налаштувань

Продовження таблиці 2.2

Клас (структура)	Опис
UserDashComponent	Клас, який містить бізнес-логіку компоненту головної сторінки користувача
CDClient	Структура, яка надає методи для взаємодії з мікросервісом неперервної доставки
CIClient	Структура, яка надає методи для взаємодії з мікросервісом неперервної інтеграції
PostgresClient	Структура, яка реалізує інтерфейс DatabaseClient, надаючи методи для взаємодії з базою даних PostgreSQL
StdoutLogger	Структура, яка реалізує інтерфейс Logger, надаючи методи для виводу логів у стандартний потік виводу
RMQQueue	Структура, яка реалізує інтерфейс Queue, надаючи методи для взаємодії з чергою повідомлень RabbitMQ
Server	Структура, що містить бізнес-логіку API мікросервісу серверної частини
RedisLock	Структура, яка реалізує інтерфейс DistLock, надаючи методи для роботи з розподіленими блокуваннями у сховищі Redis

Продовження таблиці 2.2

Клас (структура)	Опис
Kubectl	Структура, що надає методи для роботи з утилітою командного рядка kubectl
Worker	Структура, що містить бізнес-логіку мікросервісу неперервної інтеграції. Мікросервіс неперервної доставки містить аналогічну структуру

Таблиця 2.3 – Опис методів класів та інтерфейсів системи

Клас/Інтерфейс	Метод	Опис
DatabaseClient	Close()	Закриває підключення до бази даних
DatabaseClient	CreateSchema()	Створює таблиці у базі даних
DatabaseClient (API мікросервіс)	CreateUser(login, pass string, isAdmin bool)	Створює нового користувача у базі даних
DatabaseClient (API мікросервіс)	SaveUser(user *types.User)	Зберігає користувача у базі даних
DatabaseClient (API мікросервіс)	FindUser(login string)	Знаходить користувача у базі даних по логіну
DatabaseClient (API мікросервіс)	FindUserByID(userID int64)	Знаходить користувача у базі даних по ідентифікатору

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (API мікросервіс)	FindAllUsers(q url.Values)	Знаходіть усіх користувачів у базі даних
DatabaseClient (API мікросервіс)	FindAllUsersCount()	Знаходіть кількість усіх користувачів у базі даних
DatabaseClient (API мікросервіс)	ChangeUserAdminStatus(userID int64, isAdmin bool)	Змінює статус користувача на розробника або адміністратора
DatabaseClient (API мікросервіс)	CreateRepo(repo * type.GithubRepo)	Створює новий репозиторій у базі даних
DatabaseClient (API мікросервіс)	SaveRepo(repo *types.GithubRepo)	Зберігає репозиторій у базі даних
DatabaseClient (API мікросервіс)	FindRepoByName(fullName string)	Знаходить репозиторій у базі даних по назві
DatabaseClient (API мікросервіс)	FindRepoByID(repoID int64)	Знаходить репозиторій у базі даних по ідентифікатору
DatabaseClient (API мікросервіс)	DeleteRepoByID(repoID int64)	Видяляє репозиторій з бази даних

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (API мікросервіс)	FindAllUserRepos(userID int64, q url.Values)	Знаходить усі репозиторії що належать користувачеві
DatabaseClient (API мікросервіс)	FindAllUserReposCount(userID int64)	Знаходить кількість репозиторіїв що належать користувачеві
DatabaseClient (API мікросервіс)	FindAllRepos(q url.Values)	Знаходить усі репозиторії у базі даних
DatabaseClient (API мікросервіс)	CreateBranchConfig(c *types.BranchConfig)	Створює нове налаштування гілки репозиторію у базі даних
DatabaseClient (API мікросервіс)	FindBranchConfig(repoID int64, branch string)	Знаходить налаштування гілки репозиторію у базі даних
DatabaseClient (API мікросервіс)	DeleteBranchConfig(repoID int64, branch string)	Видяляє налаштування гілки репозиторію з бази даних

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (API мікросервіс)	DeleteBranchConfigByID(configID int64)	Видяляє налаштування гілки репозиторію з бази даних за ідентифікатором
DatabaseClient (API мікросервіс)	FindAllBranchConfigs(repoID int64, q url.Values)	Знаходить налаштування гілок для вказаного репозиторію у базі даних
DatabaseClient (API мікросервіс)	FindAllBranchConfigsCount(repoID int64)	Знаходить кількість налаштувань гілок для вказаного репозиторію у базі даних
DatabaseClient (мікросервіс неперервної доставки)	CreateDeployment(kd *types.Deployment)	Створює новий деплой у базі даних
DatabaseClient (мікросервіс неперервної доставки)	DeleteDeployment(kd *types.Deployment)	Видяляє деплой з бази даних
DatabaseClient (мікросервіс неперервної доставки)	FindAllDeployments(page, limit int64)	Знаходить усі деплої у базі даних

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (мікросервіс неперервної доставки)	FindDeployment(deploymentID int64)	Знаходить деплой у базі даних по ідентифікатору
DatabaseClient (мікросервіс неперервної доставки)	FindDeploymentCount()	Знаходить кількість усіх деплоїв у базі даних
DatabaseClient (мікросервіс неперервної доставки)	FindRevision(revisionID int64)	Знаходить ревізію у базі даних за ідентифікатором
DatabaseClient (мікросервіс неперервної доставки)	FindRevisions(depoymentID, page, limit int64)	Знаходить усі ревізії деплою у базі даних
DatabaseClient (мікросервіс неперервної доставки)	FindRevisionsCount(deploymentID int64)	Знаходить кількість усіх ревізій деплою у базі даних
DatabaseClient (мікросервіс неперервної доставки)	SaveDeployment(kd *types.Deployment)	Зберігає деплой у базі даних

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (мікросервіс неперервної інтеграції)	CreateBuild(b *types.Build)	Створює побудову у базв даних
DatabaseClient (мікросервіс неперервної інтеграції)	FindAllBuilds(repoID int64, branch string, page, limit int64)	Знаходить усі побудови репозиторію у базі даних
DatabaseClient (мікросервіс неперервної інтеграції)	FindBuildsCount(repoID int64, branch string)	Знаходить кількість усіх побудов репозиторію у базі даних
DatabaseClient (мікросервіс неперервної інтеграції)	CreateBuildArtifact(b *types.BuildArtifact)	Створює артефакт побудови у базі даних
DatabaseClient (мікросервіс неперервної інтеграції)	FindBuildArtifact(buildID int64)	Знаходить артефакт побудови у базі даних
DatabaseClient (мікросервіс неперервної інтеграції)	FindBuildArtifactByID(ID int64)	Знаходить артефакт побудови у базі даних за ідентифікатором

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DatabaseClient (мікросервіс неперервної інтеграції)	FindBuildByID(buildID int64)	Знаходить побудову у базі даних за ідентифікатором
DatabaseClient (мікросервіс неперервної інтеграції)	FindBuildArtifactsCount(repoID int64, branch string)	Знаходить кількість усіх побудов у базі даних
Queue	Close()	Закрити з’єднання з чергою повідомлень
Queue (мікросервіс API)	PublishCIJob(jobMsg *commonTypes.CIJob)	Надіслати повідомлення для запуску процесу інтеграції
Queue (мікросервіс неперервної доставки)	MakeCDMsgChan()	Створити канал для отримання повідомлень для запуску процесу доставки
Queue (мікросервіс неперервної інтеграції)	PublishCDJob(jobMsg *commonTypes.CDJob)	Надіслати повідомлення для запуску процесу доставки
Queue (мікросервіс неперервної інтеграції)	MakeCIMsgChan()	Створити канал для отримання повідомлень для запуску процесу інтеграції

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Logger	Info(I string)	Записати інформаційне повідомлення у лог
Logger	Error(msg string, err error)	Записати повідомлення про помилку у лог разом з її деталями
Logger	Fatal(msg string, err error)	Записати повідомлення про помилку у лог разом з її деталями, потім завершити виконання програми
DistLock	Lock(resName string)	Отримати блокування ресурсу
DistLock	Unlock(resName string)	Звільнити блокування ресурсу
BranchConfigListComponent	getPage(page: number)	Завантажити сторінку списку налаштувань гілок
BranchConfigListComponent	deleteConfig(branch: string)	Видалити налаштування
BuildComponent	loadBuild()	Завантажити побудову

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
UsersListComponent	ngOnInit()	Викликається при завантаженні компоненту
UsersListComponent	getPage(page: number)	Завантажити сторінку списку користувачів
UsersListComponent	changeAdminStatus(userID: number, admin: boolean)	Змінити статус користувача на адмінстратора чи розробника
BuildsListComponent	getPage(page: number)	Завантажити сторінку списку побудов
CreateBranchComponent	onBranchAdd()	Створити налаштування гілки репозиторію
CreateDeploymentComponent	ngOnInit()	Викликається при завантаженні компоненту
CreateDeploymentComponent	loadRepos()	Завантажити репозиторії
CreateDeploymentComponent	loadRepoArtifacts(i: string)	Завантажити артефакти побудов репозиторію
CreateDeploymentComponent	selectArtifact(i: string)	Обрати артефакт побудови

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
CreateDeploymentComponent	onDeploymentCreate()	Створити новий деплой
CreateRepoComponent	onRepoAdd()	Додати новий репозиторій
DeploymentComponent	ngOnInit()	Викликається при завантаженні компоненту
DeploymentComponent	loadDeployment()	Завантажити деплой
DeploymentComponent	loadRepo()	Завантажити репозиторій
DeploymentComponent	selectArtifact(artifactID: number)	Обрати артефакт побудови
DeploymentComponent	loadRepoArtifacts()	Завантажити артефакти побудови
DeploymentComponent	onScale()	Масштабувати деплой
DeploymentComponent	onChangeImage()	Змінити образ контейнеру деплою
DeploymentComponent	onChangeManifest()	Змінити маніфест деплою
DeploymentComponent	onDeploymentDelete()	Видалити деплой
DeploymentsListComponent	ngOnInit()	Викликається при завантаженні компоненту

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
DeploymentsListComponent	getPage(page: number)	Завантажити сторінку списку деплоїв
LoginComponent	onRegister()	Регістрація нового користувача
LoginComponent	onLogin()	Авторизація користувача
RepoComponent	ngOnInit()	Викликається при завантаженні компоненту
RepoComponent	loadRepo()	Завантажити інформацію про репозиторій
RepoComponent	onRepoDelete()	Видалити репозиторій
ReposListComponent	ngOnInit()	Викликається при завантаженні компоненту
ReposListComponent	getPage(page: number)	Завантажити сторінку списку репозиторіїв
RevisionComponent	ngOnInit()	Викликається при завантаженні компоненту
RevisionComponent	loadRevision()	Завантажити інформацію про ревізію

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
RevisionsListComponent	ngOnInit()	Викликається при завантаженні компоненту
RevisionsListComponent	getPage(page: number)	Завантажити сторінку списку ревізій
ClipperService	login(username: string, password: string)	Запит до API на авторизацію
ClipperService	register(username: string, password: string)	Запит до API на регістрацію користувача
ClipperService	addRepo(fullName: string)	Запит до API на додання репозиторію
ClipperService	getUsers(page:number, limit: number)	Запит до API на отримання інформації про користувачів
ClipperService	changeUserAdminStatus(userID: number, admin: boolean)	Запит до API на зміну ролі користувача
ClipperService	getRepos(page: number, limit: number)	Запит до API на отримання інформації про репозиторії

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
ClipperService	getRepo(repoID: number)	Запит до API на отримання інформації про репозиторій
ClipperService	deleteRepo(repoID: number)	Запит до API на видалення репозиторію
ClipperService	getBuild(buildID: number)	Запит до API на отримання інформації про побудову
ClipperService	getBuilds(repoID: number, branch: string, page: number, limit: number)	Запит до API на отримання інформації про побудови
ClipperService	addBranchConfig(branch: string, repoID: number)	Запит до API на додання налаштування гілки
ClipperService	deleteBranchConfig(repoID: number, branch: string)	Запит до API на видалення налаштування гілки

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
ClipperService	getBranchConfigs(repoID: number, page: number, limit: number)	Запит до API на отримання інформації про налаштування гілок репозиторію
ClipperService	getDeployment(depID: number)	Запит до API на отримання інформації про деплой
ClipperService	getArtifacts(repoID: number, branch: string, page: number, limit: number)	Запит до API на отримання інформації про артефакти побудов
ClipperService	addDeployment(dep: Clipper.PostDeploymentRequest)	Запит до API на створення деплою
ClipperService	deleteDeployment(depID: number)	Запит до API на видалення деплою
ClipperService	getRevisions(depID: number, page: number, limit: number)	Запит до API на отримання інформації про ревізії деплоїв

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
ClipperService	scaleDeployment(depID: number, replicas: number)	Запит до API на масштабування деплою
ClipperService	changeDeploymentImage(depID: number, imageID: number)	Запит до API на зміну образу контейнеру деплою
ClipperService	changeDeploymentManifest(depID: number, manifest: string)	Запит до API на зміну маніфесту деплою
ClipperService	getRevision(revID: number)	Запит до API на отримання інформації про ревізію деплою
StorageService	setToken(token: string)	Запам`ятати токен JWT
StorageService	parseToken(token: string)	Десеріалізувати дані з токену JWT
StorageService	isTokenExpired()	Перевірити чи актуальний токен
StorageService	isUserAdmin()	Перевірити чи є поточний користувач адміністратором

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
StorageService	getUserName()	Отримати ім’я поточного користувача.
SettingsComponent	ngOnInit()	Викликається при завантаженні компоненту
SettingsComponent	onWebhookSave()	Зберегти секрет для вебхуку
SettingsComponent	onAccessTokenSave()	Зберегти токен доступу до Github status API
CDClient	CreateDeployment(d *types.DeploymentMessage)	RPC виклик для створення деплою
CDClient	GetDeployment(deploymentID int64)	RPC виклик для отримання інформації про деплой
CDClient	GetAllDeployments(params types.PaginationQueryParams)	RPC виклик для отримання інформації про всі деплої
CDClient	DeleteDeployment(deploymentID int64)	RPC виклик для видалення деплою
CDClient	UpdateImage(d *types.DeploymentMessage)	RPC виклик для оновлення образу контейнера деплою

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
CDClient	ScaleDeployment(d *types.DeploymentMessage)	RPC виклик для масштабування деплою
CDClient	UpdateManifest(d *types.DeploymentMessage)	RPC виклик для оновлення маніфесту деплою
CDClient	GetRevision(revisionID int64)	RPC виклик для отримання інформації про ревізію деплою
CDClient	GetRevisions(deploymentID int64, params types.PaginationQueryParams)	RPC виклик для отримання інформації про всі ревізії деплою
CIClient	GetBuild(buildID int64)	RPC виклик для отримання інформації про побудову репозиторію
CIClient	GetBuildArtifact(buildID int64)	RPC виклик для отримання інформації про артефакт побудови

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
CIClient	GetAllBuilds(repoID int64, params types.BuildsQueryParams)	RPC виклик для отримання інформації про всі побудови репозиторію
CIClient	GetAllArtifacts(repoID int64, params types.BuildsQueryParams)	RPC виклик для отримання інформації про всі артефакти побудов репозиторію
Server	Run()	Запуск API серверу.
Server	Routes()	Налаштування обробників шляхів API
Server	getBuildArtifactHandler(c *gin.Context)	Обробник запитів на отримання артефакту побудов
Server	getAllArtifactsHandler(c *gin.Context)	Обробник запитів на отримання усіх артефактів побудов
Server	postBranchConfigHandler(c *gin.Context)	Обробник запитів на створення налаштування гілки репозиторію
Server	getAllBranchConfigsHandler(c *gin.Context)	Обробник запитів на отримання усіх налаштування гілки репозиторію

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Server	deleteBranchConfigHandler(c *gin.Context)	Обробник запитів на видалення налаштування гілки репозиторію
Server	getBuildHandler(c *gin.Context)	Обробник запитів на отримання побудови репозиторію
Server	postDeploymentHandler(c *gin.Context)	Обробник запитів на створення деплою
Server	getDeploymentHandler(c *gin.Context)	Обробник запитів на отримання деплою
Server	getAllDeploymentsHandler(c *gin.Context)	Обробник запитів на отримання усіх деплоїв
Server	deleteDeploymentHandler(c *gin.Context)	Обробник запитів на видалення деплою
Server	changeDeploymentImageHandler(c *gin.Context)	Обробник запитів на зміну образу контейнера деплою
Server	scaleDeploymentHandler(c *gin.Context)	Обробник запитів на масштабування деплою
Server	updateManifestHandler(c *gin.Context)	Обробник запитів на зміну маніфесту деплою

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Server	postRepoHandler(c *gin.Context)	Обробник запитів на додання репозиторію
Server	getRepoHandler(c *gin.Context)	Обробник запитів на отримання інформації про репозиторій
Server	getAllReposHandler(c *gin.Context)	Обробник запитів на отримання інформації про всі репозиторії
Server	getAllUserRepos(userID int64, c *gin.Context)	Обробник запитів на отримання усіх репозиторіїв користувача
Server	deleteRepoHandler(c *gin.Context)	Обробник запитів на видалення репозиторію
Server	getRevisionsHandler(c *gin.Context)	Обробник запитів на отримання усіх ревізій деплою
Server	getRevisionHandler(c *gin.Context)	Обробник запитів на отримання ревізії деплою

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Server	loginHandler(c *gin.Context)	Обробник запитів на авторизацію користувача
Server	registerHandler(c *gin.Context)	Обробник запитів на регістрацію користувача
Server	setSecretHandler(c *gin.Context)	Обробник запитів на встановлення секрету вебхуку користувача
Server	setAccessTokenHandler(c *gin.Context)	Обробник запитів на встановлення токену доступу до Github status API користувача
Server	webhookHandler(c *gin.Context)	Обробник вебхуків від сервісу Github
Server	jwtMiddleware(secret []byte)	Проміжний обробник, що не дозволяє виконати обробку запиту без авторизації
Server	userIsAdminMiddleware(c *gin.Context)	Проміжний обробник, що не дозволяє виконати обробку запиту якщо користувач не є адміністратором

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Server	getRevisionsHandler(c *gin.Context)	Обробник запитів на отримання усіх ревізій деплою
Server	getRevisionHandler(c *gin.Context)	Обробник запитів на отримання ревізії деплою
Kubectl	ScaleDeployment(deployment string, replicas int64)	Масштабувати деплой
Worker	Run()	Запустити мікросервіс
Worker (мікросервіс неперервної доставки)	updateDeploymentImage(dep types.Deployment, artifactID int64)	Змінити образ контейнеру деплою
Worker (мікросервіс неперервної доставки)	scaleDeployment(dep types.Deployment, replicas int64)	Масштабувати деплой
Worker (мікросервіс неперервної доставки)	executeCDJob(CDJob commonTypes.CDJob)	Виконати процес доставки
Worker (мікросервіс неперервної доставки)	updateImageFromProto(d types.Deployment)	Обробник RPC виклику зміни образу контейнеру деплою
Worker (мікросервіс неперервної доставки)	scaleFromProto(d types.Deployment)	Обробник RPC виклику масштабування деплою
Worker (мікросервіс неперервної доставки)	deleteDeployment(d types.Deployment)	Видалення деплою

Продовження таблиці 2.3

Клас/Інтерфейс	Метод	Опис
Worker (мікросервіс неперервної доставки)	initDeployment(d types.Deployment)	Ініціалізація деплою.
Worker (мікросервіс неперервної доставки)	reInitDeployment(d types.Deployment, manifest string)	Повторна ініціалізація (сброс) деплою
Worker (мікросервіс неперервної доставки)	updateManifestFromProto(d types.Deployment)	Обробник RPC виклику зміни маніфесту деплою
Worker (мікросервіс неперервної доставки)	deleteFromProto(d types.Deployment)	Обробник RPC виклику видалення деплою
Worker	startConsuming()	Запуск головного циклу обробки мікросервісу
Worker (мікросервіс неперервної інтеграції)	executeBuilder(payload types.BuilderPayload)	Запуск побудови образу контейнера

2.2 Аналіз безпеки даних

Паролі користувачів у базі зберігаються у вигляді результату обробки функцією bcrypt, тому праобраз паролю майже неможливо знайти без повного перебору, який буде сповільнено через особливість роботи bcrypt що змушує витрачати багато часу на визначення хешу паролю.

Інформація що поступає у вебхуці перевіряється за допомогою секретного ключа, що використовується для HMAC, завдяки чому перевіряється що запит був саме від GitHub.

Взаємодія з Google Container Registry та kubernetes відбувається за допомогою сервісних акаунтів яким можна обмежувати права та які самі по собі не дають повного доступу що не можна відкликати.

2.3 Висновки по розділу

У даному розділі було розроблено архітектуру мікросервісів комплексу задач. Обрано операційну систему, що найкраще підходить у якості платформи для розробки. Було обрано найкращу мову для написання мікросервісів та веб інтерфейсу, вирішено метод та протокол обміну інформацією між мікросервісами. Обрано архітектурні паттерни. Описано інтерфейси, класи, структури даних комплексу та описано їх методи. Проаналізовано безпеку даних у комплексі.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Документацію з тестування програмного забезпечення наведено в документі «Програма та методика тестування» дипломного проекту (КПІ.ІП-5201.045430.04.51)

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для повного розгортання даного комплексу задач потрібно виконати наступні етапи:

- створити дані авторізації для кластеру kubernetes;
- створити дані авторизації для реестру Google Container Registry;
- встановити мікросервіс неперервної доставки;
- встановити мікросервіс неперервної доставки;
- встановити мікросервіс API.

4.1.1 Створення даних авторизації для кластеру

Використовуючи утиліту gcloud під’єднатися до кластеру [6], після чого виконати команду на отримання даних авторизації у кластері gcloud container clusters get-credentials CLUSTER_NAME, де CLUSTER_NAME - назва кластеру. Після цього необхідні для роботи з кластером дані будуть записані у файл kubeconfig.

4.1.2 Створення даних авторизації для реестру Google Container Registry

Для створення даних авторизації потрібно перейти на головну сторінку керування Google Cloud Platform (Рисунок 4.1), після чого перейти на сторінку IAM – Сервісні аккаунти [7].

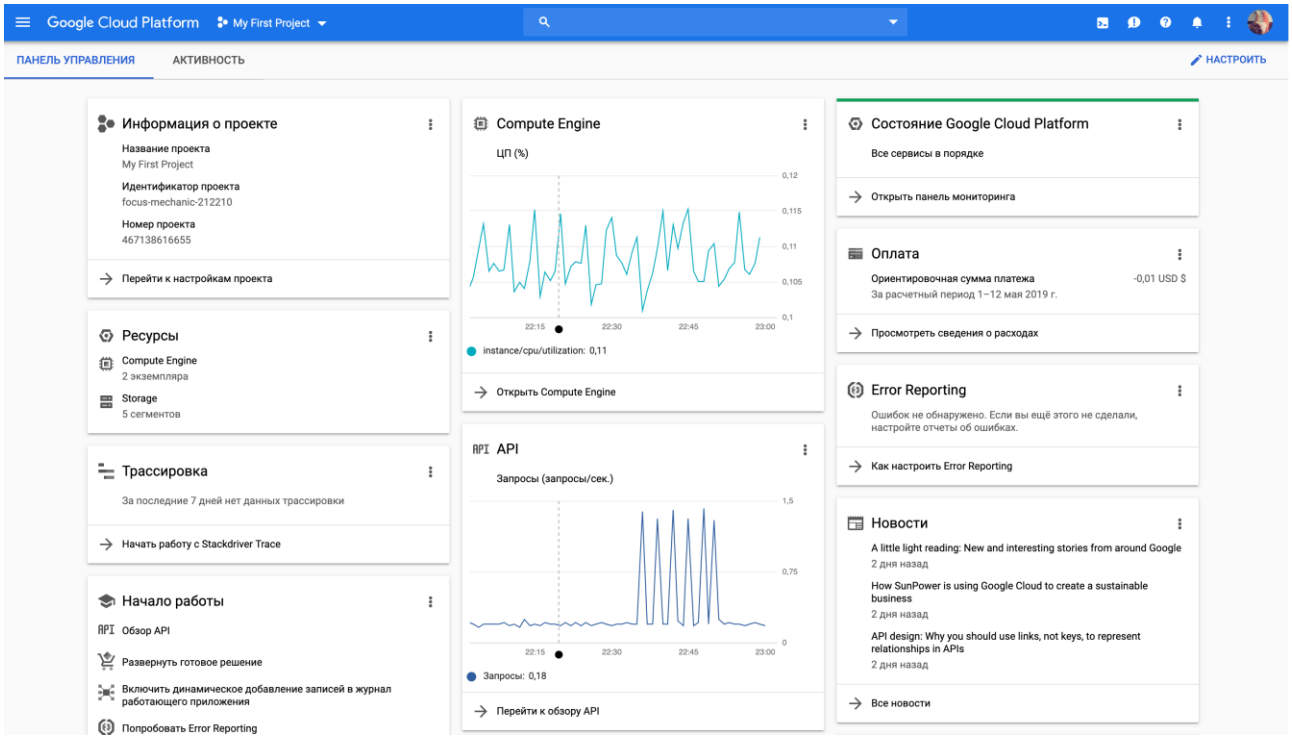


Рисунок 4.1 – Головна сторінка керування Google Cloud Platform

Далі потрібно натиснути кнопку “Створити сервісний аккаунт” та дотримуватися вказівок на екрані (Рисунок 4.2). Обрати тип файлу ключа для завантаження – JSON. Завантажити файл.

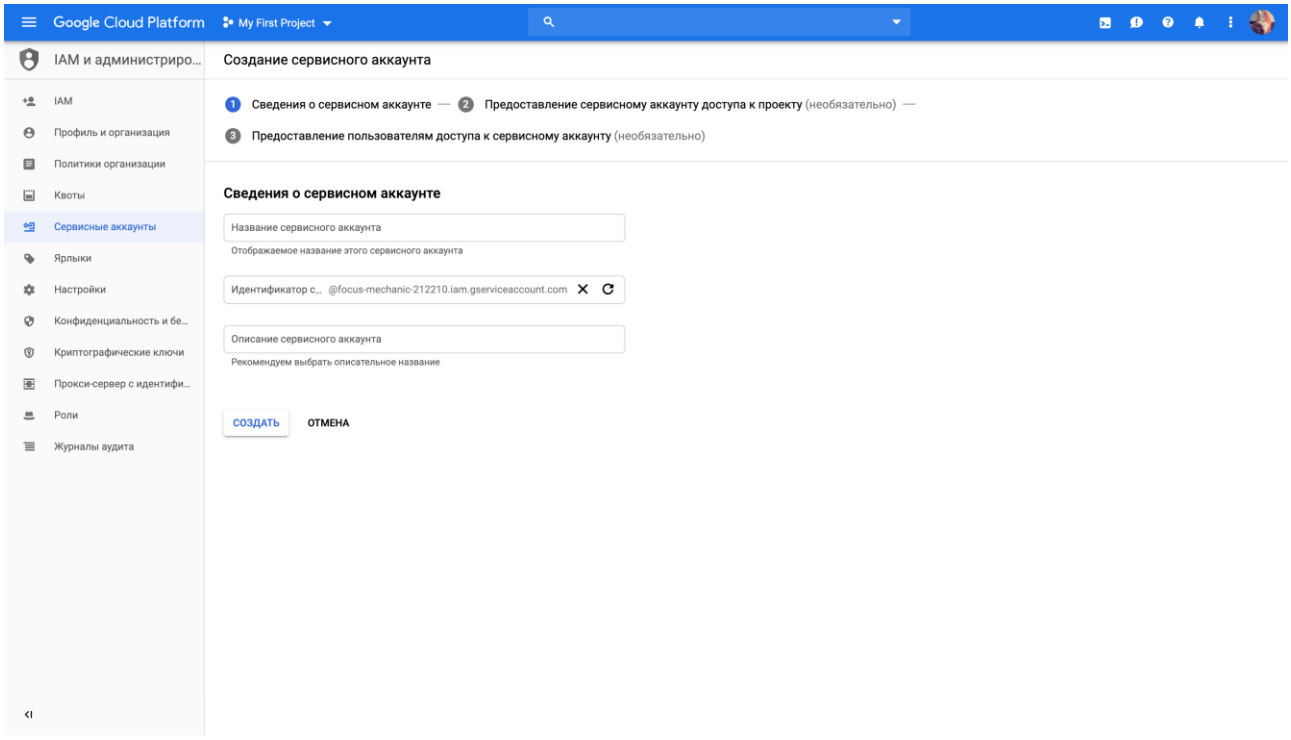


Рисунок 4.2 – Сторінка створення сервісного аккаунту для реєстру

4.1.3 Встановлення мікросервісу неперервної доставки

Для встановлення мікросервісу неперервної доставки бажано використати метод запуску за допомогою контейнеру Docker. Для цього потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-cd-worker* [8]. Після цього можна запустити мікросервіс виконавши команду *docker run clipper-cd-worker –v /PATH_TO_KUBECTL:/config/.kube --rabbitmq=RABBITMQ_ADDR –redis=REDIS_ADDR* де PATH_TO_KUBECTL – шлях до файлу авторизації у кластері отриманому у пункті 4.1.1, RABBITMQ_ADDR – адреса з'єднання з чергою повідомлень rabbitmq, REDIS_ADDR – адреса для з'єднання з redis.

4.1.4 Встановлення мікросервісу неперерної інтеграції

Встановлення мікросервісу неперервної інтеграції аналогічне до попереднього пункту. Спочатку потрібно створити образ контейнеру зі скриптом-будівником, виконавши у директорії сирцевого коду під назвою ci-builder команду *docker build . –t clipper-ci-builder*. Після цього потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-ci-worker*. Після цього можна запустити мікросервіс виконавши команду *docker run clipper-ci-worker –v /PATH_TO_JSON:/opt/clipper-ci-secrets/ --rabbitmq=RABBITMQ_ADDR –gcr=GCR_URL –builder=BUILDER_IMAGE –redis=REDIS_ADDR* де PATH_TO_JSON – шлях до файлу авторизації у Google Container Registry отриманому у пункті 4.1.2, RABBITMQ_ADDR – адреса з'єднання з чергою повідомлень rabbitmq, REDIS_ADDR – адреса для з'єднання з redis, GCR_URL – адреса регіону Google Container Registry для використання, BUILDER_IMAGE – назва або повна адреса образу контейнеру з будівником.

4.1.5 Встановлення мікросервісу API

Встановлення мікросервісу API виконується аналогічно. Потрібно у директорії з сирцевим кодом виконати команду *docker build . –t clipper-api*.

Після цього можна запустити мікросервіс виконавши команду *docker run clipper-api --rabbitmq=RABBITMQ_ADDR –ci=CI_ADDR –cd=CD_ADDR* де RABBITMQ_ADDR – адреса з’єднання з чергою повідомлень rabbitmq, CI_ADDR – адреса для з’єднання з мікросервісом неперервної інтеграції, CD_ADDR – адреса для з’єднання з мікросервісом неперервної доставки.

4.2 Інструкція користувача

Інструкцію адміністратора програмного забезпечення наведено в документі «Керівництво користувача» дипломного проекту (КПІ.ІП-5201.045430.11.34)

4.3 Інструкція адміністратора

Інструкцію адміністратора програмного забезпечення наведено в документі «Керівництво адміністратора» дипломного проекту (КПІ.ІП-5201.045430.12.33)

ВИСНОВКИ

У розділі “Аналіз вимог до програмного забезпечення” було описано та проаналізовано предметну область розробки.

У розділі “Моделювання та конструювання програмного забезпечення” було розроблено архітектуру мікросервісів комплексу задач.

У розділі “Аналіз якості та тестування програмного забезпечення” було розроблено план тестування комплексу задач та виконано приклад тестування.

У розділі “Впровадження та супровід програмного забезпечення” було описано процес встановлення комплексу задач на апаратну платформу. Розроблено інструкції для користувача-програміста та для адміністратора.

У рамках даного дипломного проєкту було застосовано набуті знання з розробки баз даних, архітектури програмного забезпечення, веб розробки, розробки програм з використанням паралельних обчислень, безпеки даних, об’єктно-орієнтованого програмування, мультипарадигменого програмування.

Розроблений комплекс задач є повноцінним програмним продуктом, готовим для використання, який легко може бути доповнений новим функціоналом та масштабований.

ПЕРЕЛІК ПОСИЛАНЬ

1) CircleCI [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://circleci.com.

2) Jenkins [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://jenkins.io.

3) Helm [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://helm.sh.

4) deploy-node-app [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://github.com/kubesail/deploy-node-app.

5) TEST PLAN OUTLINE IEEE829 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.fit.vutbr.cz/study/cources/ITS/public/ieee829.html.

6) Google Cloud Platform https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl – Configuring cluster access for kubectl (Налаштування доступу до кластеру за допомогою утиліти kubectl) Google Cloud Platform – Configuring cluster access for kubectl (Налаштування доступу до кластеру за допомогою утиліти kubectl) [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl.

7) Додаткові методи аутентіфікації Google Cloud Platform [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://cloud.google.com/container-registry/docs/advanced-authentication#json_key_file

8) Опис утиліти командного рядку Docker [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.docker.com/engine/reference/commandline/.

9) Документація оркестратору Kubernetes [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://kubernetes.io/docs/home/.

10) Документація мови Go [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://golang.org/doc/.

11) Документація мови Typescript [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/home.html.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud Platform

CD-ROM

(Вид носія даних)

10 арк, 1488 Кб

(Обсяг програми (документа) , арк.,) Кб)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-admin-dash',
  templateUrl: './admin-dash.component.html',
  styleUrls: ['./admin-dash.component.css']
})
export class AdminDashComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}

import { Component, OnInit, Input } from '@angular/core';
import { Router } from '@angular/router';
import { Clipper } from '../services/types';
import { ClipperService } from '../services/clipper.service';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/do';

@Component({
  selector: 'branch-configs-list',
  templateUrl: './branch-configs-list.component.html',
  styleUrls: ['./branch-configs-list.component.css']
})
export class BranchConfigsListComponent implements OnInit {
  branchConfigs: Observable<Array<Clipper.BranchConfig>>;
  p = 1;
  total: number;
  loading: boolean;
```

```
@Input() properties: any = {repoID: 0};

constructor(
  public clipper: ClipperService,
  private router: Router,
) {
}

ngOnInit() {
  this.getPage(1);
}

getPage(page: number) {
  this.loading = true;
  this.clipper.getBranchConfigs(this.properties.repoID,
    page, 10)
  .subscribe(res => {
    if (!res.err) {
      this.branchConfigs = Observable.of(res.configs)
      .do(resObs => {
        this.loading = false;
        this.total = res.total;
        this.p = page;
      });
    }
  });
}
deleteConfig(branch: string) {
  console.log(branch);
  this.clipper.deleteBranchConfig(this.properties.repoID, branch)
  .subscribe(res => {
    if (!res.err) {
      this.getPage(this.p);
    }
```

```
    })
  }
}

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Clipper } from '../services/types';
import { ClipperService } from '../services/clipper.service';

@Component({
  selector: 'app-build',
  templateUrl: './build.component.html',
  styleUrls: ['./build.component.css']
})
export class BuildComponent implements OnInit {
  buildID: number;
  build: Clipper.Build;

  constructor(
    public clipper: ClipperService,
    private route: ActivatedRoute
  ) { }

  ngOnInit() {
    this.route.params.subscribe(params => {
      this.buildID = +params['id'];
      this.loadBuild();
    });
  }
  loadBuild() {
    this.clipper.getBuild(this.buildID)
    .subscribe(res => {
      this.build = res;
    });
```

```
  }
}

import { Component, OnInit, Input } from '@angular/core';
import { Clipper } from '../services/types';
import { ClipperService } from '../services/clipper.service';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';
import 'rxjs/add/operator/do';

@Component({
  selector: 'builds-list',
  templateUrl: './builds-list.component.html',
  styleUrls: ['./builds-list.component.css']
})
export class BuildsListComponent implements OnInit {
  builds: Observable<Array<Clipper.Build>>;
  p = 1;
  total: number;
  loading: boolean;
  @Input() properties: any = {repoID: 0, branch: 'master'};

  constructor(
    public clipper: ClipperService,
  ) {
  }

  ngOnInit() {
    this.getPage(this.p);
  }

  getPage(page: number) {
    this.loading = true;
    this.clipper.getBuilds(this.properties.repoID,
```

```
        this.properties.branch, page, 10)
      .subscribe(res => {
       if (!res.err) {
        this.builds = Observable.of(res.builds)
        .do(resObs => {
         this.loading = false;
         this.total = res.total;
         this.p = page;
        });
       }
      });
    }

}

package CDApi

import (
        "context"

        "github.com/revan730/clipper-api/log"
        "github.com/revan730/clipper-api/types"
        commonTypes "github.com/revan730/clipper-common/types"
        "google.golang.org/grpc"
)

type CDClient struct {
        gClient commonTypes.CDAPIClient
        log     log.Logger
}

func NewClient(address string, logger log.Logger) *CDClient {
        conn, err := grpc.Dial(address, grpc.WithInsecure())
        if err != nil {
```

```
                logger.Fatal("Couldn't connect to CD gRPC", err)
        }

        c := commonTypes.NewCDAPIClient(conn)
        client := &CDClient{
                gClient: c,
                log:    logger,
        }
        return client
}

func (c *CDClient) CreateDeployment(d *types.DeploymentMessage) error {
        protoMsg := types.ProtoFromDeploymentMsg(d)
        _, err := c.gClient.CreateDeployment(context.Background(), protoMsg)
        return err
}

func (c *CDClient) GetDeployment(deploymentID int64) (*commonTypes.Deployment,
error) {
        protoMsg := &commonTypes.Deployment{
                ID: deploymentID,
        }
        return c.gClient.GetDeployment(context.Background(), protoMsg)
}

func (c *CDClient) GetAllDeployments(params types.PaginationQueryParams)
(*commonTypes.DeploymentsArray, error) {
        return c.gClient.GetAllDeployments(context.Background(),
        &commonTypes.DeploymentsQuery{
                Page:   int64(params.Page),
                Limit:  int64(params.Limit),
        })
}
```

```
func (c *CDClient) DeleteDeployment(deploymentID int64) error {
	protoMsg := &commonTypes.Deployment{
		ID: deploymentID,
	}
	_, err := c.gClient.DeleteDeployment(context.Background(), protoMsg)
	return err
}

func (c *CDClient) UpdateImage(d *types.DeploymentMessage) error {
	protoMsg := types.ProtoFromDeploymentMsg(d)
	_, err := c.gClient.ChangeImage(context.Background(), protoMsg)
	return err
}

func (c *CDClient) ScaleDeployment(d *types.DeploymentMessage) error {
	protoMsg := types.ProtoFromDeploymentMsg(d)
	_, err := c.gClient.ScaleDeployment(context.Background(), protoMsg)
	return err
}

func (c *CDClient) UpdateManifest(d *types.DeploymentMessage) error {
	protoMsg := types.ProtoFromDeploymentMsg(d)
	_, err := c.gClient.UpdateManifest(context.Background(), protoMsg)
	return err
}

func (c *CDClient) GetRevision(revisionID int64) (*commonTypes.Revision, error) {
	protoMsg := &commonTypes.Revision{
		ID: revisionID,
	}
	return c.gClient.GetRevision(context.Background(), protoMsg)
}
```

```
func (c *CDClient) GetRevisions(deploymentID int64, params types.PaginationQueryParams) (*commonTypes.RevisionsArray, error) {
	return c.gClient.GetRevisions(context.Background(),
	&commonTypes.RevisionsQuery{
		DeploymentID: deploymentID,
		Page:   int64(params.Page),
		Limit:  int64(params.Limit),
	})
package CIApi

import (
	"context"

	"github.com/revan730/clipper-api/types"
	"github.com/revan730/clipper-api/log"
	commonTypes "github.com/revan730/clipper-common/types"
	"google.golang.org/grpc"
)

type CIClient struct {
	gClient commonTypes.CIAPIClient
	logger  log.Logger
}

func NewClient(address string, logger log.Logger) *CIClient {
	conn, err := grpc.Dial(address, grpc.WithInsecure())
	if err != nil {
		logger.Fatal("Couldn't connect to CI gRPC", err)
	}

	c := commonTypes.NewCIAPIClient(conn)
	client := &CIClient{
		gClient: c,
		logger:  logger,
```

```
        }
        return client
    }

    func (c *CIClient) GetBuild(buildID int64) (*commonTypes.Build, error) {
        return c.gClient.GetBuild(context.Background(),
            &commonTypes.Build{ID: buildID})
    }

    func (c *CIClient) GetBuildArtifact(buildID int64) (*commonTypes.BuildArtifact, error) {
        return c.gClient.GetBuildArtifact(context.Background(),
            &commonTypes.BuildArtifact{BuildID: buildID})
    }

    func (c *CIClient) GetAllBuilds(repoID int64, params types.BuildsQueryParams)
(*commonTypes.BuildsArray, error) {
        return c.gClient.GetAllBuilds(context.Background(),
            &commonTypes.BuildsQuery{RepoID: repoID,
                Branch: params.Branch,
                Page:   int64(params.Page),
                Limit:  int64(params.Limit),
            })
    }

    func (c *CIClient) GetAllArtifacts(repoID int64, params types.BuildsQueryParams)
(*commonTypes.ArtifactsArray, error) {
        return c.gClient.GetAllArtifacts(context.Background(),
        &commonTypes.BuildsQuery{RepoID: repoID,
            Branch: params.Branch,
            Page:   int64(params.Page),
            Limit:  int64(params.Limit),
        })
    }
```

```
package cmd

import (
	"fmt"
	"os"

	"github.com/revan730/clipper-api/src"
	"github.com/revan730/clipper-api/log"
	"github.com/revan730/clipper-api/types"
	"github.com/spf13/cobra"
)

var (
	logVerbose bool
	serverPort int
	dbAddr     string
	db         string
	dbUser     string
	dbPass     string
	adminLogin string
	adminPass  string
	jwtSecret  string
	rabbitAddr string
	ciAddr     string
	cdAddr     string
)

var rootCmd = &cobra.Command{
	Use:   "clipper-api",
	Short: "REST API microservice of Clipper CI\\CD",
}
```

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ О.А. Павлов
“___” _____ 2019 р.

Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud

Програма та методика тестування

КПІ.ІП-5201.045430.03.51

“ПОГОДЖЕНО”
Керівник проекту:
_____ О.А. Халус

Нормоконтроль:
_____ К.І. Ліщук

Виконавець:
_____Є.О. Бурлаченко

Київ – 2019 року

ЗМІСТ

1 АНАЛІЗ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Етап тестування є важливим в процесі розробки даного продукту у зв`язку з великою складністю системи. Також продукт передбачає постійну взаємодію із зовнішніми онлайн сервісами.

Тестування програмного забезпечення — техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином.

Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення й забезпечення якості програмного забезпечення, до якого належать всі складові ділового процесу, а не тільки тестування.

Зазвичай, поняття якості обмежується такими поняттями як коректність, надійність, практичність, безпечність, але може містити більше технічних вимог, котрі описані у стандарті ISO 9126. Склад та зміст супутньої документації процесу тестування визначається стандартом IEEE 829—1998 Standard for Software Test Documentation [5]. Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів — це по суті дослідницький та творчий процес, а не тільки створення та виконання рутинної процедури.

План тестування ПЗ включає в себе обсяг, підхід, ресурси та план усіх методів тестування. План описує об’єкти та функції що будуть протестовані, тип тестів, ресурси та план необхідний для виконання тестування.

У рамках цього плану буде виконано тестування частини продукту, що відповідає за процес інтегрування програмного коду продукту, починаючи від налаштування та закінчуючи завантаженням артефакту побудови до сервісу збереження контейнерів Docker.

У даному плані будуть протестовані наступні функції:

- реєстрація користувачів;
- авторизація користувачів;
- реєстрація репозиторіїв Github;
- налаштування для гілок репозиторіїв Github;
- налаштування секретного ключа користувача;
- створення задачі побудови репозиторію;
- виконання задачі побудови репозиторію;
- виконання задачі розгортання репозиторію;
- реєстрація розгортання репозиторію;
- реєстрація кластеру Kubernetes.

Налаштовані наступні тестові модулі:

- реєстрація користувача;
- реєстрація користувача якщо логін вже зайнято;
- реєстрація користувача з невалідним паролем;
- авторизація користувача;
- авторизація користувача з неправильним паролем;
- реєстрація репозиторію Github;
- реєстрація репозиторію Github з неправильною назвою;
- реєстрація вже наявного у системі репозиторію Github;
- ввімкнення автоінтеграції для гілки репозиторію;
- вимкнення автоінтеграції для гілки репозиторію;
- введення секретного ключа користувача;
- введення токену авторизації користувача;
- введення невалідного токену авторизації користувача;

- створення задачі побудови репозиторії через webhook;
- створення задачі побудови репозиторію з секретним ключем;
- створення задачі побудови репозиторію з невалідним секретним ключем;
- виконання задачі побудови репозиторію;
- виконання задачі побудови репозиторію для приватного репозиторію;
- виконання задачі побудови репозиторію що має помилки.

2 ПІДХОДИ ДО ТЕСТУВАННЯ

В рамках даного плану будуть використані наступні методи тестування:

- компонентне;
- інтеграційне;
- продуктивності.

2.1 Компонентне тестування

Методом компонентного тестування будуть перевірені логічно окремі частини API сервера та робітника, такі як:

- оброблювачі шляхів API;
- проміжкові оброблювачі шляхів API;
- методи доступу до бази даних;
- окремі допоміжні функції;

2.2 Інтеграційне тестування

Методом інтеграційного тестування будуть перевірені взаємодії між модулями системи, такі як:

- робота арі сервера та робітника з базою даних;
- робота арі сервера та робітника з чергою подій;
- взаємодія робітника та будівельника;
- взаємодія робітника та сервісу GitHub;
- взаємодія будівельника та регістру контейнерів;
- ланцюг процесу інтеграції репозиторію, від отримання запиту на інтеграцію репозиторію до завантаження контейнеру у реєстр.

2.3 Тестування продуктивності

Методом тестування продуктивності буде перевірена швидкодія наступних елементів системи:

- запити до бази даних;
- обмін повідомленнями між мікросервісами;
- паралельне виконання декількох процесів побудови репозиторію.

3 КРИТЕРІЇ ПРОХОДЖЕННЯ ТЕСТУВАННЯ

3.1 Компонентне тестування

Для компонентного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один пункт не був успішно виконаний – тестування вважається не пройденим.

3.2 Інтеграційне тестування

Для інтеграційного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один пункт не був успішно виконаний – тестування вважається не пройденим.

3.3 Тестування швидкодії

Для тестування швидкодії критерієм проходження є успішне виконання тесту з кожним доступним набором параметрів (кількість даних у одному запиті, кількість запитів, конкурентність) не довше ніж максимально допустимий час. У разі якщо хоча б один варіант тесту не був успішним або виконувався довше максимально допустимого часу – тестування вважається не пройденим.

4 ПРОЦЕС ТЕСТУВАННЯ

4.1 Дані до тестів

Вхідними даними для компонентного тестування є набори параметрів на яких очікується певний результат, що є вихідними даними даного тесту.

Вхідними даними для інтеграційного тестування є набори повідомлень що будуть передані від одного компоненту системи до іншого відповідно до конкретного тесту. Вихідними даними для даного виду тестування є результат роботи останнього компоненту у ланцюзі (наприклад, запис у базі даних у випадку тестування взаємодії API серверу та бази даних).

Вхідними даними до тестування швидкодії є набори даних, що покривають усі варіанти роботи системи у конкретному випадку. Вихідними даними є швидкість обробки запитів, кількість оброблених запитів, кількість неправильних реакцій на набір даних, дані по навантаженню на апаратну платформу (завантаженість процесору, вільна оперативна пам’ять, завантаженість мережі тощо).

4.2 Задачі тесту

Кожен тест повинен перевірити як правильність програми у відповідності до умов виконання тесту (test-driven development), так і виявити можливі помилки у роботі.

4.3 План виконання

Компонентне тестування повинне виконуватися до інтеграційного, яке, у свою чергу, виконується до тестування швидкодії.

5 ВИМОГИ ДО СЕРЕДОВИЩА

5.1 Апаратна частина

Вимоги до апаратної частини співпадають з вимогами з технічного завдання.

5.2 Програмна частина

Для виконання тестування апаратна платформа повинна мати операційну систему на базі Linux або іншу unix-сумісну, зі встановленою системою контейнеризації Docker.

5.3 Вимоги до безпеки

Для виконання тестування бажано створити окремий токен доступу до аккаунту GitHub та окремий сервісний аккаунт системи GCP.

5.4 Інструменти

Для виконання тестування використовувати наступні програмні інструменти:

- go test (Входить до набору розробки мови Go);
- Postman;
- GitHub;
- GCP;
- Vegeta (https://github.com/tsenart/vegeta) – для тестування швидкодії.

6 ОПИС КОНТРОЛЬНОГО ПРИКЛАДУ

У якості прикладу можна навести тестування логіну користувача за допомогою утиліти Postman.

Вхідні дані:

- пара логін та пароль *notexistant notexistant* (не наявні у комплексі);
- пара логін та пароль *admin wrongpassword* (неправильний пароль, існуючий користувач);
- пара логін та пароль *admin admin* (існуючий користувач).

Вихідні дані:

- повідомлення про помилку у вигляді JSON стрічки *{“err”: “Failed to login”}* ;
- повідомлення з токеном доступа при успішному логіні у вигляді JSON стрічки *{ “token”: “TOKEN”}* де TOKEN – токен доступу до API.

У даному тестовому прикладі усього 3 ситуації, перші 2 відповідають першим двом парам вхідних даних та першому набору вихідних, відповідно 3 ситуація – останньому набору вхідних та вихідних даних.

Для виконання тесту потрібно в Postman створити POST запит за адресою http://API_URL/api/v1/login де API_URL - адреса API мікросервісу.

У якості тіла запиту використати JSON об’єкт наступного формату:

{“login”: “LOGIN”, “password”: “PASSWORD”} де LOGIN та PASSWORD – логін та пароль з однієї з пар вхідних даних. Після введення даних натиснути кнопку “Send” та перевірити відповідність вихідних даних до наведених вище (Рисунок 6.1).

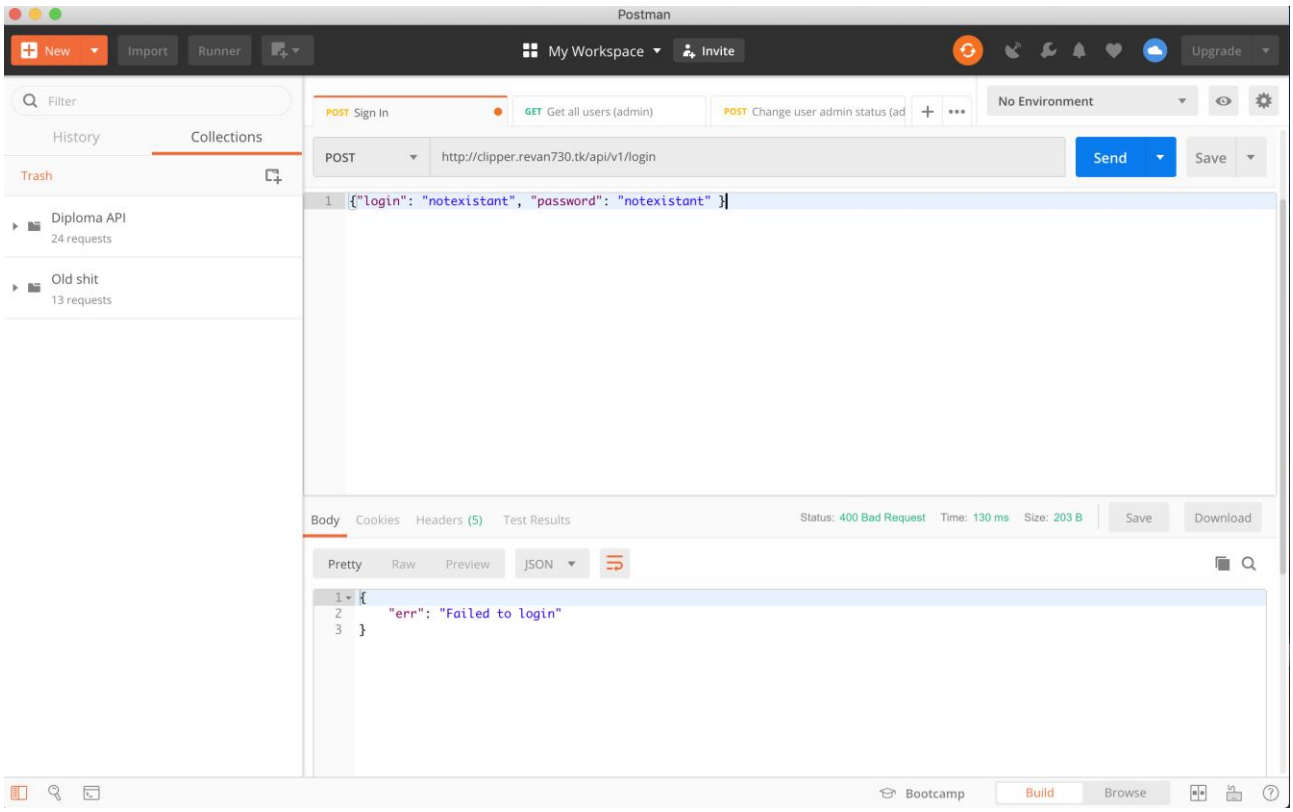
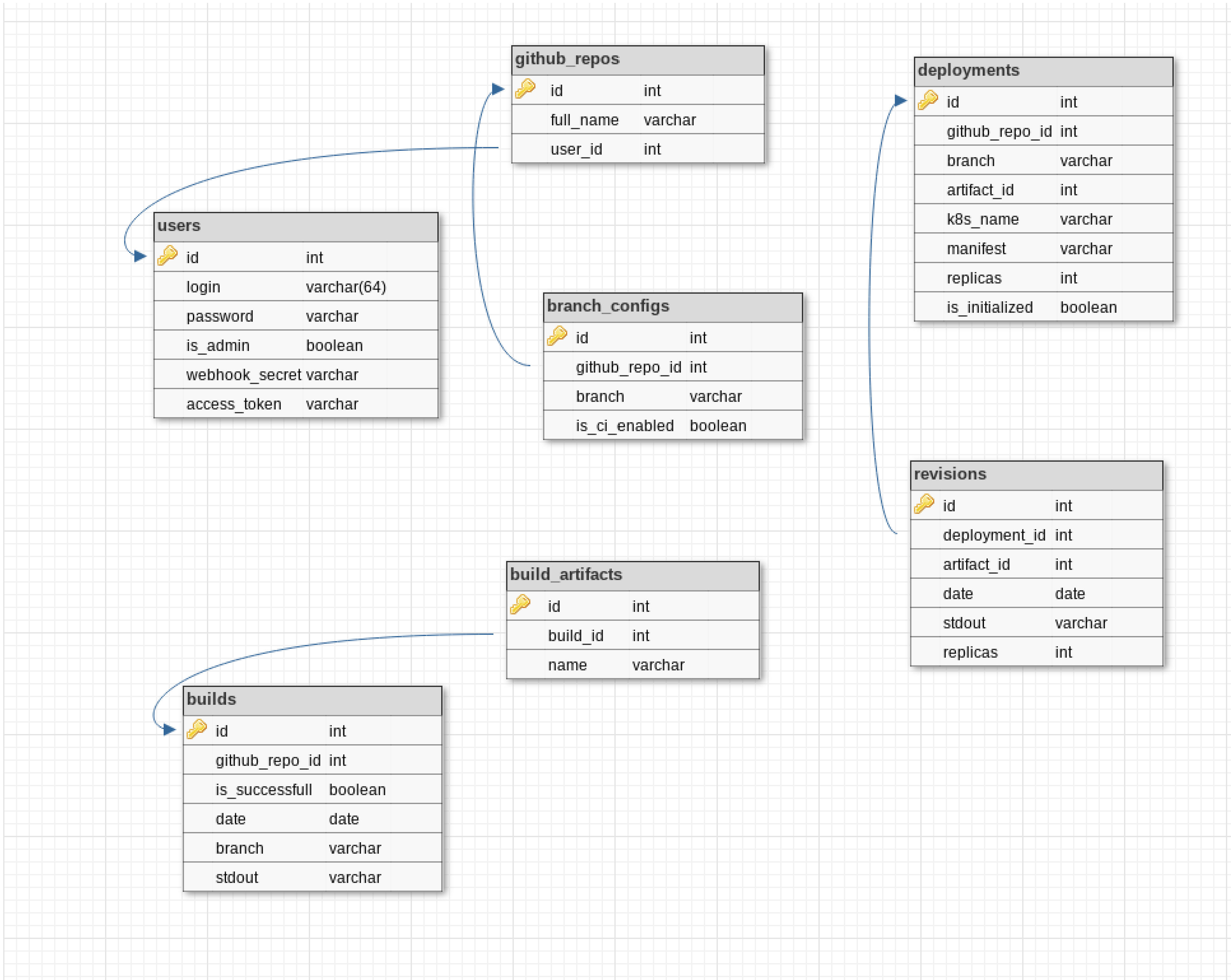


Рисунок 6.1 – Приклад виконання тесту

					КПІ.ІП-5201.045430.05.99		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використання	Літера	Маса / Масштаб
Розробив		Бурлаченко Є.О					
						Аркуш 1	Аркушів 1
Перевірив		Халус О.А					
Т. кон.					Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-52	
Н. кон.		Ліщук К.І					
Затвердив		Халус О.А					

КПІ.ІП-5201.043430.05.99



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Бурлаченко Є.О.		
Перевірив		Халус О.А.		
Т. кон.				
Н. кон.		Ліщук К.І.		
Затвердив		Халус О.А.		

КПІ.ІП-5201.043430.05.99

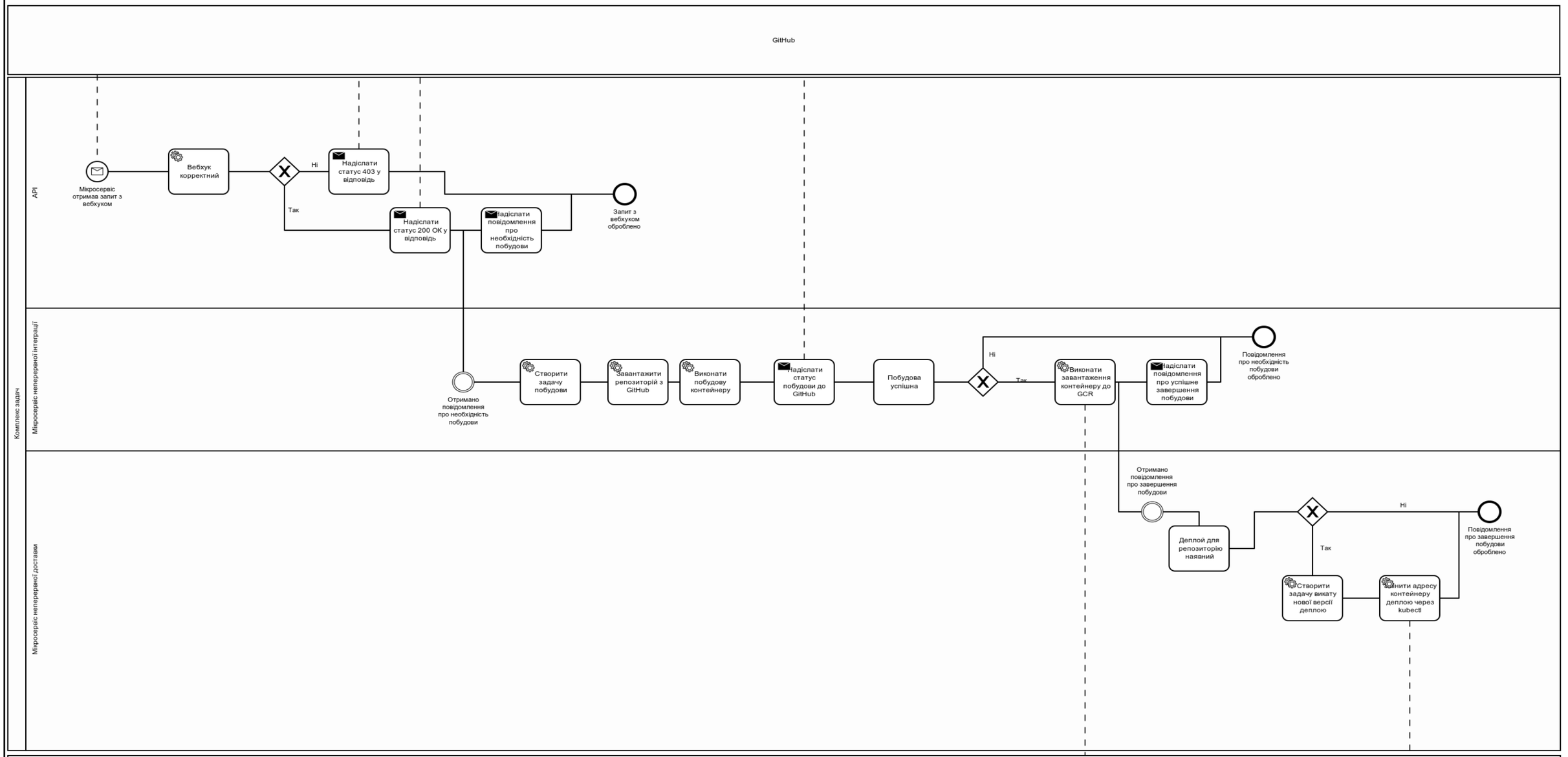
Схема структурна бази даних

Літера	Маса	Масштаб

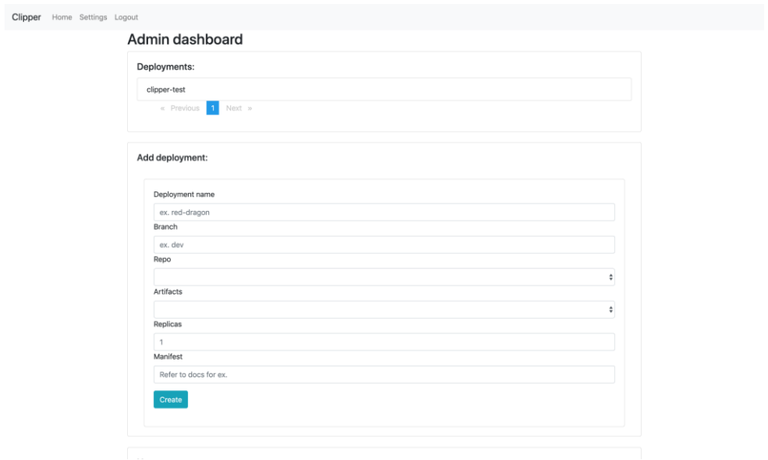
Аркуш 1 | Аркушів 1

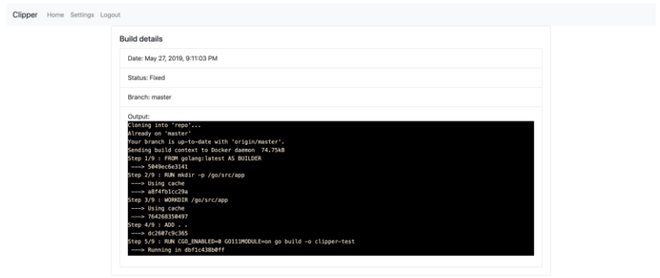
Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud

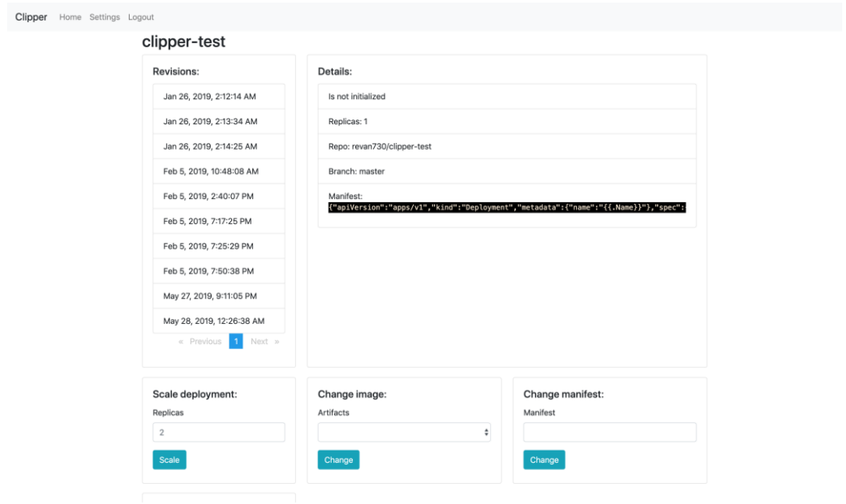
КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-52

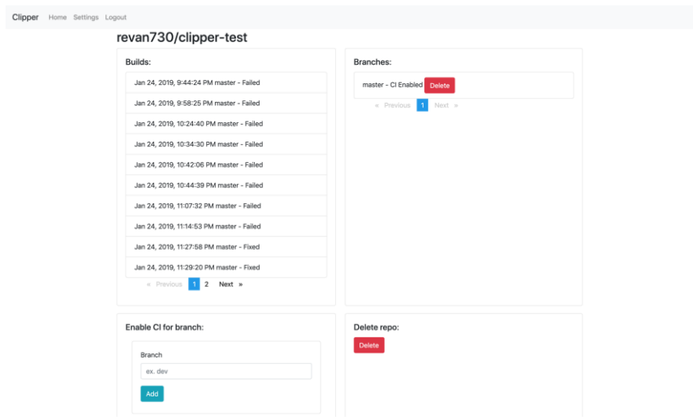


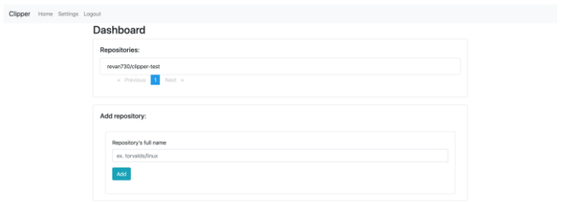
					КПІ.ІП-5201.043430.05.99		
						Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна бізнес процесу*		
Розробив		Бурлаченко Є.О.				Аркуш 1	Аркушів 1
Перевірив		Халус О.А.					
Т. кон.					*Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-52*	
Н. кон.		Ліщук К.І.					
Затвердив		Халус О.А.					











					КПІ.ІП-5201.045430.05.99	
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна екранних форм*	Літера / Маса / Масштаб
Розробив		Бурлаченко Є.О.				
Перевірив		Халус О.А.				Аркуш 1 / Аркушів 1
Т. кон.						
					Комплекс задач розгортання програмних продуктів з використанням платформи Google Cloud	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІП-52*
Н. кон.		Ліщук К.І.				
Затвердив		Халус О.А.				